

5. L'adressage direct

Il concerne, tout comme l'adressage étendu, le contenu d'un octet de la mémoire mais il ne peut être utilisé que s'il s'agit d'un octet dont l'adresse est inférieure à 256. Un exemple :

```
LDAA < $80
```

Après exécution de cette instruction, il y aura dans le registre A le même nombre que celui qui est écrit à l'adresse \$80 (128 décimal).

```
PRINT PEEK (128) ; réponse : 34.
```

C'est donc la valeur 34 qui a été placée dans A.

Il n'y a rien d'autre à ajouter sur ce mode d'adressage que d'ailleurs l'on n'emploie qu'épisodiquement. Remarquons tout de même que l'adresse de l'octet est donnée en hexadécimal et qu'elle est précédée du signe < .

Nous en avons terminé avec ce chapitre, le cap difficile de la théorie est passé. Asseyons-nous devant Alice et voyons comment nous allons lui faire comprendre puis exécuter un programme écrit en assembleur.

4

ÉTUDE D'UN EXEMPLE

Retrouvons le programme BASIC qui nous a servi, dans le Chapitre 2, à mettre en évidence les qualités graphiques de notre ordinateur.

```

10 POKE 48929 , 65 : REM   LETTRE A
20 POKE 48930 , 1  : REM   MODE ALPHA
30 POKE 48931 , 23 : REM   ROUGE/BLANC
40 POKE 48934 , 27 : REM   LIGNE N° 20
50 POKE 48935 , 27 : REM   COL. N° 27
60 POKE 48936 , 1  : REM   EXECUTION

```

Inutile de le retaper, il n'est présenté ici que pour nous remettre en tête la marche à suivre quand on veut afficher un caractère sur l'écran. Cette marche à suivre, nous allons le voir, est rigoureusement la même en BASIC et en assembleur.

LE MODE ÉDITEUR-ASSEMBLEUR

Respectons les consignes du manuel d'utilisation :

```

CLEAR 100 , 18000   puis ENTER
&                  puis ENTER

```

Dès que ces deux ordres ont été tapés, l'écran s'efface et le fond devient bleu. Nous sommes maintenant sous le contrôle de l'éditeur-assembleur. Le microprocesseur 6803 attend nos ordres. Allons-y sans crainte.

```

1      ORG    $4A00    ; 18944 DECIMAL
2      EXC    DEBUT
3  DEBUT  LDAA    #$41    ; CODE ASCII A
4      STAA   $BF21    ; 48929 DECIMAL
5      LDAA   #$1
6      STAA   $BF22    ; 48930 DECIMAL
7      LDAA   #$17    ; 23 DECIMAL
8      STAA   $BF23    ; 48931 DECIMAL
9      LDAA   #$19    ; 27 DECIMAL
10     STAA   $BF26    ; 48934 DECIMAL

```

```

11      STAA   $BF27    ; 48935 DECIMAL
12      LDAA   #$1
13      STAA   $BF28    ; 48936 EXECUTION
14  FIN    BRA     FIN

```

Voici quelques indications sur les précautions à prendre pour entrer ce programme :

- N'indiquez surtout pas, comme en BASIC, les numéros des lignes que vous voyez ici écrits à gauche. Nous ne les avons fait apparaître que parce que cela vous permettra de vous y retrouver plus facilement dans les explications.
- Sachez que tout ce qui suit un point-virgule est ignoré par l'ordinateur ; le point-virgule est en assembleur l'équivalent de l'instruction BASIC REM.
- Vous devez écrire les mots DEBUT et FIN (que l'on appelle des étiquettes) à partir du bord gauche de l'écran.
- Enfin, n'oubliez jamais de laisser au moins deux espaces libres avant d'écrire une instruction et d'en laisser au moins un entre les différents constituants d'une ligne.

ANALYSE DU PROGRAMME

```

ORG    $4A00    (ligne 1)

```

On s'est bien compris, n'est-ce pas ? Vous avez laissé au moins deux espaces avant de taper les lettres O, R et G et vous avez ensuite laissé au moins un espace avant d'afficher \$4A00 ? Et vous n'avez mentionné aucun numéro de ligne ? Bien, alors voici à quoi correspond cette ligne :

La directive ORG apparaît toujours à la première ligne des programmes. Elle donne à l'ordinateur des indications sur la façon dont il devra ranger, dans sa mémoire, les codes machine. Ne cherchons pas, pour l'instant, à y comprendre quoi que ce soit. D'une part parce que nous aurons l'occasion d'y revenir et d'autre part parce que cela n'a pas de rapport direct avec ce que nous sommes en train d'analyser.

```

EXC    DEBUT    (ligne 2)

```

Le programme, à proprement parler, n'a pas encore commencé ; EXC est une directive qui va préciser à la machine à quelle ligne se trouve la première instruction réelle du programme. En l'occurrence, c'est de la ligne 3 qu'il est question, ligne à laquelle nous avons *collé une étiquette*. Au lieu de l'appeler "ligne 3", nous l'appelons "ligne DEBUT". Et, du coup, EXC ligne 3 devient EXC DEBUT. Naturellement, nous aurions pu remplacer le mot DEBUT par ABCDE (ou n'importe quel assemblage de cinq lettres maximum). Il aurait fallu alors faire débiter la ligne 3 par ABCDE.

DEBUT LDAA #\$41 (ligne 3)

A partir de là, nous intervenons sur le microprocesseur lui-même. LDAA est une instruction que nous avons déjà rencontrée. Elle signifie que le registre A va contenir la valeur hexadécimale \$41 (65 décimal). Le symbole # est là pour indiquer le mode d'adressage immédiat. On ignore quel nombre se trouvait dans l'accumulateur avant cette instruction, mais maintenant on est sûr de la valeur de A : c'est 65 en décimal (\$41).

STAA \$BF21 (ligne 4)

STAA est une instruction très fréquemment utilisée en assembleur : elle signifie que la valeur contenue dans A va devoir être inscrite dans un octet de la mémoire. STAA est l'abréviation de STORE A qui, en anglais, veut dire RANGER A. Le mode d'adressage choisi, l'étendu, nous laisse entendre que \$BF21 (48929 décimal) est l'adresse d'un octet de la mémoire. En définitive, c'est dans cet octet que sera rangé le contenu de A. Cette instruction est donc l'équivalent assembleur de la ligne BASIC 10 POKE 48929,65 car, ne le perdons pas de vue, le registre A contient le nombre 65.

Voilà reconstituée, sous sa forme assembleur, la ligne BASIC 10. Il y a donc, pour l'instant, le code ASCII de la lettre A dans l'octet 48929. Avant de poursuivre notre étude, tirons la leçon de ce que l'on vient de faire : pour écrire une valeur dans un octet, on place cette valeur dans le registre A (LDAA en mode immédiat) puis on la range en mémoire (STAA en mode étendu).

LDAA #\$1 (ligne 5)
STAA \$BF22 (ligne 6)

Nous pouvons aller plus vite maintenant que le principe est compris. On écrit dans l'accumulateur le nombre 1 et on le transfère dans l'octet \$BF22 (48930 décimal). On vient, par là-même, de porter son choix sur le type d'affichage alphanumérique standard.

LDAA #\$17 (ligne 7)
STAA \$BF23 (ligne 8)

Puisque \$17 vaut 23 en décimal, ces deux lignes ont pour but d'écrire la valeur 23 dans l'octet \$BF23 (48931 décimal). La lettre majuscule A apparaîtra donc en rouge sur fond blanc.

LDAA #\$19 (ligne 9)
STAA \$BF26 (ligne 10)

Nous continuons à respecter l'enchaînement des lignes BASIC : voici les deux instructions qui correspondent à POKE 48934,27. Le caractère sera visible sur la ligne numéro 20 car, mais vous ne l'avez pas oublié, les lignes 1, 2 ... 7 sont sautées par l'ordinateur.

STAA \$BF27 (ligne 11)

Aucun ordre LDAA n'ayant été programmé, la valeur contenue par A est restée la même. Cette ligne fournit à l'ordinateur la dernière chose qui lui manquait : le numéro de la colonne sur laquelle se verra la lettre A.

LDAA #\$1 (ligne 12)
STAA \$BF28 (ligne 13)

Il ne reste plus qu'à rendre effectif l'affichage : ceci se fait en écrivant le nombre 1 dans l'octet \$BF28 (48936 décimal).

FIN BRA FIN (ligne 14)

Nous utilisons tous dans nos programmes BASIC des lignes du genre :

100 GOTO 100

Nous avons sous les yeux la forme équivalente en assembleur. BRA est une instruction de branchement inconditionnel et comme ce branchement s'effectue à la ligne courante, le programme boucle sans fin. Bien entendu, l'intérêt de cet état de choses sera de nous laisser le temps de voir ce qui va apparaître sur le téléviseur : la lettre A coloriée en rouge sur fond blanc.

EXÉCUTION DU PROGRAMME

Le programme est entièrement tapé, il ne reste plus qu'à commander à l'ordinateur de l'exécuter.

Pressez à la fois les touches CTRL et 1. Le message suivant apparaît en bas de l'écran :

LISTING (ECRAN , IMP , ENTER) ?

Appuyez sur E puis sur la touche ENTER. Vous devez voir apparaître la totalité de votre programme avec, en plus, divers renseignements. Nous avons, devant nous, ce que l'on appelle le listing d'assemblage.

1			ORG	\$4A00
2			EXC	DEBUT
3	4A 00	86 41	DEBUT LDAA	#\$41
4	4A 02	B7 BF 21	STAA	\$BF21
5	4A 05	86 01	LDAA	#\$1
6	4A 07	B7 BF 22	STAA	\$BF22
7	4A 0A	86 17	LDAA	#\$17
8	4A 0C	B7 BF 23	STAA	\$BF23
9	4A 0F	86 19	LDAA	#\$19
10	4A 11	B7 BF 26	STAA	\$BF26
11	4A 14	B7 BF 27	STAA	\$BF27
12	4A 17	86 01	LDAA	#\$1
13	4A 19	B7 BF 28	STAA	\$BF28
14	4A 1C	20 F2 FIN	BRA	FIN

0 ERREUR(S) PASSE 1

0 ERREUR(S) PASSE 2

SYMBOLES :

DEBUT = 4A00 FIN = 4A1C

FICHIER OBJET ?

A la dernière ligne, l'ordinateur demande si l'on désire sauvegarder le programme objet sur une cassette. Appuyez sur BREAK. Apparaît alors le message qui nous intéresse :

EXECUTION IMMEDIATE ?

Répondez en enfonçant la touche 0 (puis ENTER). Le programme est enfin exécuté et cela se traduit par le dessin de la lettre majuscule A, en rouge sur blanc, en bas et à droite de l'écran.

Voilà, nous sommes arrivés à nos fins ; le premier pas est franchi. En fin de compte, programmer en assembleur n'est pas plus compliqué qu'en BASIC. C'est simplement un peu plus long car le microprocesseur ne comprend que des actions élémentaires ; il faut lui *mâcher le travail*.

N'oubliez pas que le programme boucle indéfiniment dans la ligne 14. La touche BREAK ne vous sera d'aucune utilité pour reprendre Alice en main. Il n'y a qu'un seul moyen pour sortir d'un programme assembleur qui boucle : enfoncer la touche INIT placée sur la face arrière de la machine.

Laissez la fin de ce chapitre de côté. Dans un premier temps tout au moins, ce qui suit peut très bien être ignoré. Inutile donc de vous encombrer l'esprit avec des notions qui, en définitive, ne nous concernent pas directement. Vous reprendrez le livre à cette page quand vos connaissances se seront un peu stabilisées.

LE LANGAGE MACHINE

Un ordinateur ne comprend que des nombres et pour lui les expressions LDAA, STAA ou BRA ne veulent absolument rien dire. Il va donc falloir lui traduire le programme que nous avons écrit en assembleur sous la seule forme qui lui soit compréhensible : les codes machine.

Insistons bien sur la différence qu'il y a entre les langages assembleurs et machine : le premier est conçu pour l'esprit humain et il est

formé d'instructions qui ont un sens pour nous. Quand on écrit LDAA, on sait très bien ce qui se passera dans le processeur, on n'a pas besoin de faire un gros effort pour comprendre que le contenu du registre A sera modifié et remplacé par une nouvelle valeur. La forme assembleur permet d'écrire des programmes qui soient lisibles, des programmes qui soient constitués de mots ou d'abréviations dont on s'habituerà très vite à connaître le sens.

Quant au langage machine, il est constitué d'une série de nombres que l'ordinateur, lui, est capable d'interpréter. Il y a naturellement une correspondance absolue entre les instructions assembleur et les codes machine qui leur sont relatifs.

Livrons-nous, pour la première et pour la dernière fois, à la traduction en langage machine du programme que nous avons écrit en assembleur. C'est Alice elle-même qui se charge normalement de ce travail. Servez-vous du tableau de l'Annexe C.

3	4A 00	86 41	DEBUT	LDAA	#\$41
4	4A 02	B7 BF 21		STAA	\$BF21

86 est l'équivalent pour la machine de LDAA. Vous constatez que LDAA se code de différentes façons suivant le mode d'adressage. Celui qui nous intéresse est l'immédiat, dans la première colonne donc. Il faudra toujours se souvenir que les codes machine sont écrits en hexadécimal ; 86, ainsi que tous les autres codes de ce tableau, respecte cette règle.

41 est le nombre qui suit 86. Le microprocesseur, après avoir interprété 86, s'attendra à ce qu'on lui dise avec quel nombre il doit charger A. Puisque 41 vient à la suite de 86, il comprendra que la valeur 41 (65 décimal) doit être placée dans l'accumulateur.

B7 est le code machine de l'instruction STAA. Il doit être choisi dans la bonne colonne, celle de l'adressage étendu. C'est en effet ce mode que nous avons décidé d'utiliser en écrivant le programme assembleur. Quand l'ordinateur va lire ce code, il saura qu'il lui faut alors s'intéresser aux deux valeurs suivantes.

BF et 21 forment le nombre hexadécimal BF21. La machine, ayant rencontré B7, comprendra qu'elle doit placer le contenu du registre A dans un octet de la mémoire ; dans le numéro 48929 (\$BF21) bien sûr.

Arrêtons nos efforts de traduction, cela devient vite fastidieux. Retenons que notre programme assembleur correspond, pour le processeur, à la suite de nombres :

86, 41, B7, BF, 21, etc.

L'ensemble de ces valeurs est appelé le code machine et il est directement exécutable. Il suffit, pour le faire exécuter, de brancher le processeur sur la première de ces valeurs. C'est exactement ce qui est réalisé quand nous répondons OUI à la question EXECUTION IMMEDIATE ?

Il reste pour clore ce chapitre, à indiquer comment l'ordinateur fait pour savoir à quel endroit de sa mémoire se trouve le premier des codes machine. Il utilise, pour cela, la première indication du programme :

ORG \$4A00

Cette directive lui donne l'adresse à partir de laquelle sera rangé le code machine. Puisque \$4A00 est égal à 18944 en notation décimale, on en déduit que les nombres 86, 41, B7, etc., seront écrits dans les octets 18944, 18945, 18946, etc.

Vous pourrez vérifier ceci en étudiant les nombres qui, sur le listing d'assemblage, suivent les numéros de ligne :

3	4A 00	86 41
4	4A 02	B7 BF 21

86 est écrit dans l'octet \$4A00, B7 est écrit dans l'octet \$4A02, etc.